JIRA FOR TEST MANAGEMENT:
Bridging the Gap Between Design and Issues

QAComplete
Flexible, Enterprise Test Management for Teams

LEARN MORE ABOUT QAComplete
JIRA has become one of the most popular tools in the world – helping software teams track and manage their issues and projects in an Agile manner. For many teams and companies, JIRA has become one of the central tools used on an everyday basis to help them deliver on every release. Because of its widespread popularity, an entire ecosystem of integrations, add-ons and plug-ins has developed – all readily available on the Atlassian Marketplace.

In this evolving marketplace, businesses and individuals are designing a host of technologies to help JIRA users (as well users of other Atlassian technologies) get more out of JIRA’s existing capabilities.

Many of these technologies also solve challenges that JIRA isn’t natively set up to support (such as time keeping, advanced diagramming and charting) and offer added functionality in areas where JIRA alone might be satisfactory, but advanced capabilities may be desired by certain teams.

Test management certainly falls into this final category. Most QA and dev teams use JIRA to track issues and to manage releases. For many teams, especially small ones, native issue management in JIRA may be sufficient in terms of a test management strategy. Many teams, however, will require more advanced capabilities, particularly as it relates to linking test design issues with requirements, tests and defects. To help with this, there are two types of add-ons a QA or development team can use – a lightweight test management plug-in or an integration with a full-fledged test management tool. Each of these has their own benefits and drawbacks, just as managing your tests using only JIRA does. In this eBook, we will look at the plusses and minuses of all three approaches, with the goal of helping test and development teams choose the tool that is best for them.
What Is a Test, Exactly?

As anybody who has worked in multiple organizations can attest, terminology and definitions can vary. Because of this, it’s critical to have a clear set of definitions of what a test is composed of, in order to compare the benefits and drawbacks of different strategies and tools used to support them. The three terms we will focus on for the purposes of this eBook are:

1. Test Definition – which can be subdivided into:
   “Launching the Orders application” comment to a test. To do this:
   - Test Steps
   - Tests
   - Test Sets
2. Test Execution
3. Test History

Test Definition

On the surface, a test definition is fairly obvious - it’s how you define your test. Going a layer deeper, tests are constructed of a few different subcomponents, which we will define as steps, tests and sets.

A step is easy to understand – it is simply a single action or a single result. If a user clicks a button, for example, do they go to the next page? That would be a classic definition of a step.

A test is a little bit bigger – it’s the sum of different test steps. The button step from the previous example would be one piece a login test. If that same user logs in to an application, that would be an entire test, considering of typing in a username, a password, and clicking submit. That action alone has a minimum of three or four steps, depending on how your test is designed.

Finally, if you group these tests together, you get a test set. The aforementioned login test can be part of a larger eCommerce test set, consisting of a login test, a shopping cart test and a check out test (and more!). This test set consists of different groups of tests, which in turn consist of a series of test steps. Depending on your organization’s terminology, you may refer to each of these differently, but the most important thing is understanding that there are three levels, ranging from individual actions (steps), groups of actions (tests), and groups of tests (test sets).
Test Definition

Once you have decided what the set looks like, what the test looks like and what the steps look like, it’s time to perform the execution at the three different stages as well. For example, if you start at the step level – let’s take the for the login step for example - you might perform that action at the test level or at the set level, grouping those tests in one set and executing against that particular group.

There are different ways of executing these tests – you may perform some of these tests manually, and others might be automated. Executing tests sits at the heart of your testing efforts, but how you go about doing it depends on your strategy and how your tests are structured.

Test History

If you view test design as an input and test execution as the engine, then test reporting is the output of your QA efforts. A team’s ability to effectively communicate (and contextualize) results to assess the overall quality of the application under test relies on their ability to report on their efforts effectively. The results of your tests often depend on your execution environment. When you test across different operating systems, browsers or operate at different resolution settings, the results of your test may change. For example, using the login step example, you may have different results if you run the same login step on Chrome, Mozilla, or different operating systems such as Windows 10, Windows 7, Windows XP.

As important as reporting is, it’s even more critical to have a record of your results over time – which is where test history comes in. Since you have performed or executed these tests at three different levels, which is step level, test level, or even test set level, you need to have a history of label at these three different levels as well. Browsers, OS, combinations, and other variations are extremely important to the test execution. They’re even more important down the history because the same definition might pass or fail on different environments.
Putting it Together

When you look at each of the three layers, it becomes clear that they each serve different roles in your testing strategy. Thus, it is important to separate your test definition from test execution and test history for the following reasons:

1. Reusability – Many tools give you the ability to clone components of tests. By doing this, you save time since you don’t have to manually recreate different assets every time. If you are testing across different environments, many of your tests will be the same, so the ability to clone these steps give you the duel benefit of saving time and giving you separate histories for each of those combined environments and executions. Bringing it up a level, this can also be used to clone tests across different test sets, allowing you to easily recreate tests in different projects. The same benefits of time savings and maintaining separate test histories apply here as well.

2. Combined Manual and Automated Testing – one of the benefits of separating the definitions of your tests, steps and sets from the execution portion is that you can combine the histories together into single reportable items. For example, in a sample login test, you may execute it manually on a mobile device while automating the test in various browsers in different test environments. It’s important to track the environments, it’s important to track all those separate executions and separate histories but they all share the same definition. You want to be able to combine all of them into one test set.

3. Test History – Another important benefit is the ability to get specific detailed history at three different stages, whether it is at the test level, the step level or even at the set level. You want to see who is making changes at these three different stages, when those changes are being made, and what changes are being made. So separating these three different layers out really helps you, at the higher level, give these four different value statements, which is extremely helpful in ensuring your team is productive and you can keep reusing assets rather than actually recreating assets for each and every level.

Choices for Test Management Tools

For all its powerful capabilities, JIRA does not have test management-specific capabilities built into the product. Since JIRA is often the singular project and issue management tool for teams that use it, there are three ways...
to overcome the challenge of managing their tests while keeping their issues and defects within JIRA. The correct choice may vary based on the needs of the team, and considerations such as budget, team size, and complexity of workflow.

**Using Native JIRA Capabilities for Test Management**

For many teams, the preferred course of action is one where they don’t require any internal software. Since JIRA is very configurable to an organization’s business process, it is relatively straightforward to configure JIRA for test management purposes on top of their existing projects.

For example, using the earlier example, it’s easy to create a custom issue at the test set level. In order to maintain the three-layer hierarchy, the test and step levels can be related to each other using custom issue types related to each other as subtasks. After that, it’s as simple as setting up a custom specific workflow for each test set. With the right JIRA know-how, this customized approach makes sense for teams with the internal resources to set this up. From there, you can set up sub-tasks as your tests, and include the tests steps in custom fields (or in the description field) within the subtasks.

The benefits of this approach are pretty straightforward. From a cost standpoint, you don’t have to purchase another tool or add-on, since you’re using your existing JIRA instance. Since you’re operating within your JIRA instance, it is also easy to link to any issues or defects stored there. JIRA is an inherently customizable tool, so it is fairly easy to build a customized workflow to suit your needs. This is especially true if your team is doing mostly manual testing, since there is no need to link to any third-party automation tool. Reporting is also solid, since, by definition, you are using JIRA’s standard, out-of-the-box reports.

However, there are also several major downsides to going with this approach. Significantly, customizing JIRA for test management means that test definition, history and execution are bundled by test set, rather than stored as separate objects. This has reporting implications, but also limits reusability and create redundancies. For example, for each test you run, you need to recreate new subtasks, as execution is tied to each sub-task. Additionally, you can’t reuse elements of tests easily, causing added time in creating tests. You may also run into issues at the test step level, as using custom fields on subtasks prevents you from seeing execution status at that level. Finally, there are inherit-
ability concerns (if a test passes, mark everything below it as “pass”), as creating static tests in JIRA means you can’t pass test set status down to the test or step. This becomes especially tricky when you try to link in automated tests, since manual work will be required.

Using a 3rd Party Native JIRA Plug-in

Another fast-growing method of managing tests in JIRA is the use of native JIRA test management add-ons. These lightweight tools are typically developed by third parties, and are an obvious choice for JIRA-native teams who have outgrown manual processes in spreadsheets or via customized JIRA workflows. Since issues and defects are already in JIRA, why not use a tool to manage tests within JIRA as well?

There are some natural advantages to using these tools. Since they are native plug-ins, using the JIRA interface will minimize friction during rollout. Additionally, it’s as easy to link to issues as it would be if you built out your own home-grown solution within JIRA. Some teams might also find that there are some inherent benefits to developers and testers using the same tool. Reporting, again, is based on JIRA’s out-of-the-box reporting capabilities, so if you’re comfortable with that, then using an add-on will be sufficient for those purposes. Finally, these tools are pretty inexpensive on a per-user basis. This comes with a big caveat, as they are often priced on the number of users in a single JIRA implementation. If there are a large volume of users that don’t require a test management tool, this might net out that benefit. Thus, there is an inherent advantage for small teams using these tools, but this obviously makes it difficult to scale.

Despite these benefits, there are also some pretty significant challenges to teams who want to use these tools to do enterprise-caliber test management. We mentioned that from a pricing standpoint, it may be difficult to scale usage of these tools, but there are additional scalability problems when it comes to actually using them. Since there is no hierarchy or ability to set up folders, scaling larger scale applications becomes difficult. For example, if you are testing a web application, and a different test set corresponds with a different part of that website, it might be difficult to pinpoint problem areas. Hierarchies are easily conveyed through folder structures, less easily through things like tagging and items like that that these solutions tend to adopt.
Like with using standalone JIRA, there are reusability concerns. If you want to create a login test, for example, you still cannot reuse tests or steps across different test sets, which means these tests and steps need to be manually recreated each time. Additionally, these tools typically do not allow you to add images to steps, so rather than saying “does the page match the attached picture,” you’re forced to describe the desired outcome.

One other major shortcoming is the inability to retain historical data. For example, if a test step passes and, the following day, you want to make modifications at the step or test layer, the data becomes overwritten and your ability to make conclusions based on this data becomes compromised, since the edited test is shown in previous runs. This ties back to the topic we touched on earlier about the separation between test history and execution. In this case, they are intertwined too closely, which makes reporting a lot less valuable down the road.

Finally, there are often added costs associated with using these tools. Since JIRA does not have any built-in capabilities to perform automated tests, purchasing added third party plug-ins may be required. The same is true if you desire reporting beyond what JIRA offers out of the box. When this is coupled with the aforementioned scalability concerns, it makes sense that teams looking to evolve in terms of test management scale look to vendors that offer more full-fledged platforms.

**Using a Test Management Platform Alongside JIRA**

The third way teams can manage tests with JIRA is to use a standalone test management tool. Going with this approach typically represents the most advanced evolution of test management from an enterprise standpoint, and for many teams, carries a number of advantages over customized JIRA workflows and lightweight plugins. Despite this, a test management tool can represent a major investment for teams, especially in terms of time, thus selecting the right test management tool is key.

Before getting into the benefits and pitfalls of selecting a test management tool, let’s review what we mean by a test management tool – a separate, standalone tool used by teams to manage tests, even if your requirements and/or defects sit in JIRA. By integrating directly with JIRA, teams can get the benefits of a full-featured test management tool while maintaining their existing requirement and defect tracking used by the rest of the extended team.
But what are the benefits of using a standalone test management tool? Not surprisingly, they address some of the shortcomings of the first two methods. One of the biggest benefits is maintaining the division between test execution and history. The ability to have full traceability and versioning for your tests, sets and steps allows you to account for any changes you make to your tests, as well as easily organize and group your tests.

Another major, related advantage pertains to reusability. Since test definition, security and history are separate, it makes it much easier to reuse assets across test sets, at both the test and step level. Everything in a test management tool is reusable, at all levels, which is important when you consider that even something as simple as a login test will be tested in different environments, browsers, configurations, security roles — you name it. Not only does a test management tool make it easy to recreate those tests, the ability to easily see how that test performed in each of these different configurations is extremely powerful.

There are also some of the more enterprise-class features that larger teams might need that aren’t present in lighter-weight tools, such as easy-to-setup environment configurations, single sign-on and security privileges.

Chief among these are built-in integrations to automation tools. As we discussed earlier, one of the biggest downsides of using an add-on is that you often need to buy extra plug-ins to work with different automation frameworks. Many of the test management tools have built-in integrations tools to different integrations vendors or even open source frameworks and tools such as Selenium and SoapUI.

Despite all of these advantages, there are definitely things that teams should look out for when choosing an automation vendor. Here are some of the biggest:

- Limited JIRA Integration: If your requirements and defects are stored in JIRA, then using a standalone Test Management tool can only provide added value if it integrates seamlessly with JIRA. Be wary of tools that only offer limited integration, and seek out tools that have robust, bilateral syncs.

- Cost: Depending on the size of your team, using any test management tool (plug-in or standalone tool) will be more expensive from a cost standpoint than, say, customizing JIRA. Tools that help your team save time by reducing rework can help you save money by reducing testing cycles. Make sure you take this into account when choosing a tool.
• Customizability: In general, the more customizable the tool, the better. This is especially true if you are Agile, or have a unique business workflow. Look for tools that are flexible enough to fit your business’ process.

• Browser Compatibility: This may seem like a small deal compared to other items on this list, but a characteristic of some tools (particularly the more legacy ALM tools) is the lack of browser support. This is one little thing that drives end users crazy!

• Automation Tool Integration: With most teams using a combination of manual and automated tests, the ability to hook in your automated tests easily is paramount. Keep an eye out for tools that have natural integrations with automation tools (both open source and commercial) that you already use.

• Success and Training: This is surprisingly important to getting your team to adopt a new tool quickly and easily. Seek out vendors who offer this for a nominal cost, preferably free.
Flexible, Enterprise Test Management for Teams

QACOMPLETE

TRY IT FOR FREE